## 1. Scope, Context, Execution context , Scope Chain, Closures, Call and Apply

Reference:

[**http://ryanmorr.com/understanding-scope-and-context-in-javascript/**](http://ryanmorr.com/understanding-scope-and-context-in-javascript/)

[**http://www.cnblogs.com/yuzhongwusan/archive/2012/03/08/2385396.html**](http://www.cnblogs.com/yuzhongwusan/archive/2012/03/08/2385396.html)

**lexical scoping**: in JavaScript, the scope of a variable is defined by its location within the source code (it is apparent lexically) and nested functions have access to variables declared in their outer scope.

**Execution context**is a concept in the language spec that—in layman's terms—roughly equates to the 'environment' a function executes in; that is, variable scope (and the *scope chain*, variables in closures from outer scopes), function arguments, and the value of the this object.

**Hoisting :** function **declarations** are "hoisted" to the top of their scope.

a();

b();

function a() { }

var b = function() { }

The call to a() will succeed because its declaration was hoisted to the top; a was assigned to automatically before program execution began. The call to b() will fail with a TypeError because b will not be defined until line 4.

Interpreted:

Var a;

a=function(){};

a();

Interpreted

var b;

b();

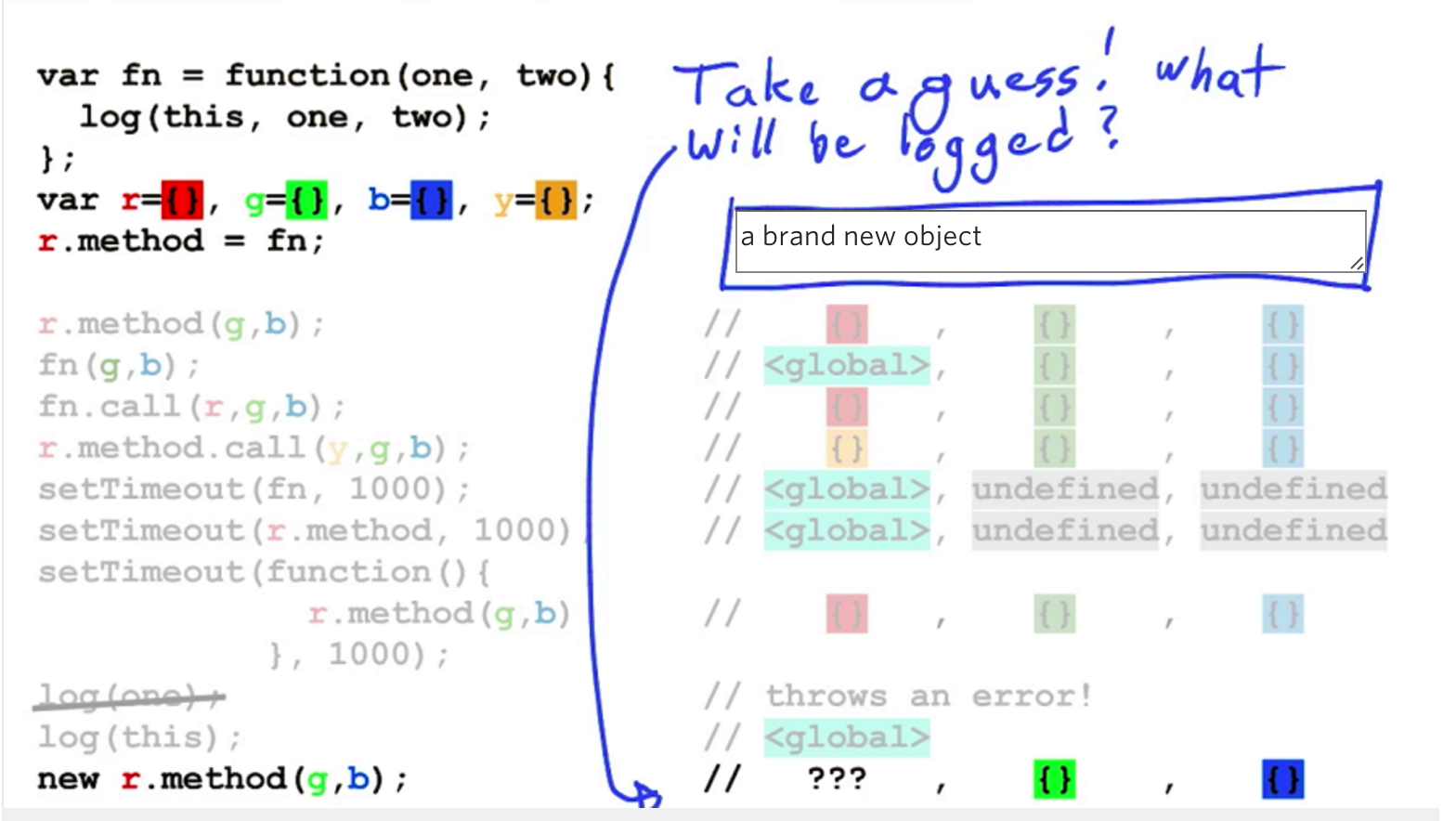
b = function() {}

// Uncaught TypeError: b is not a function, so the result is undefined.

2.

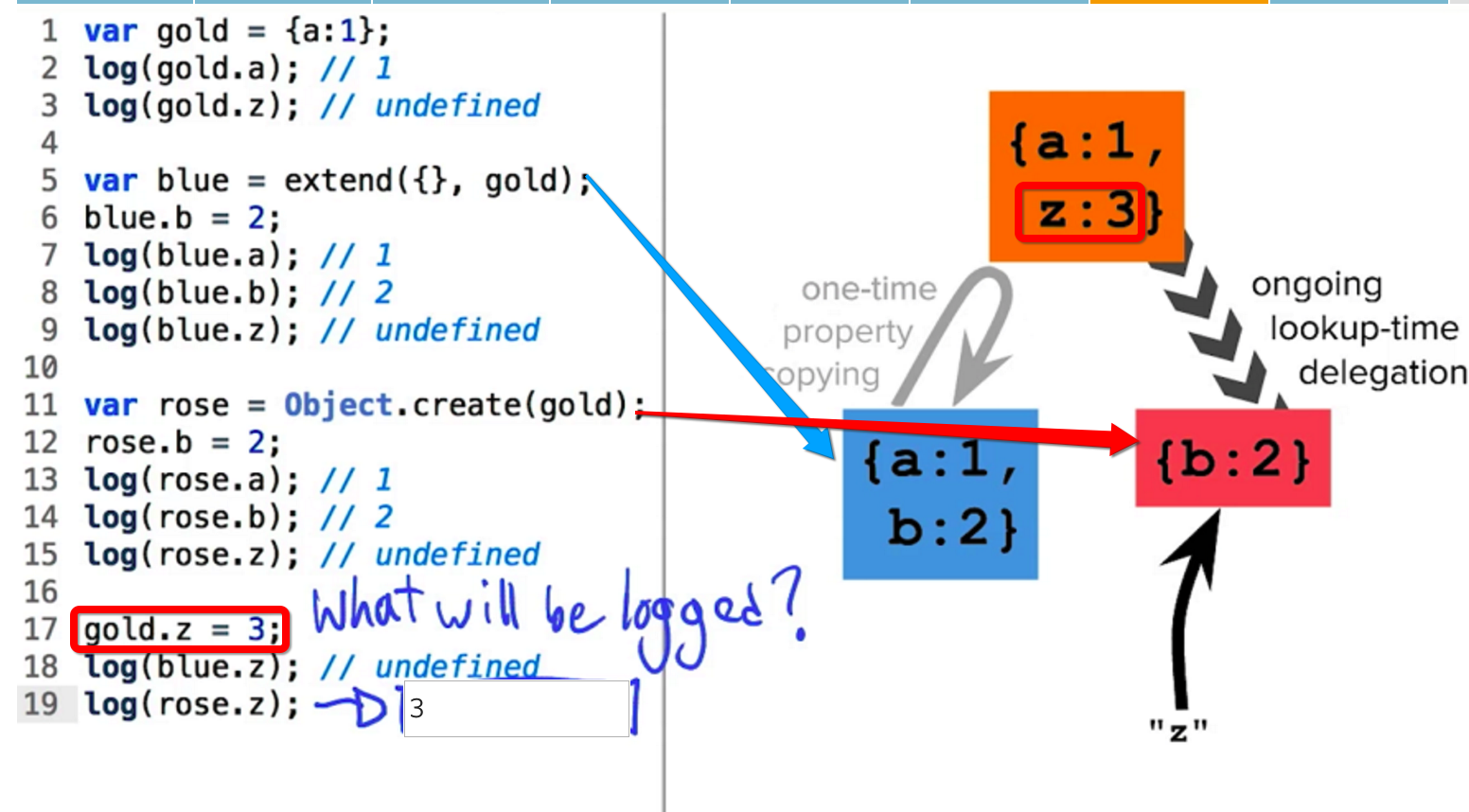
reference from Udacity

key word : this



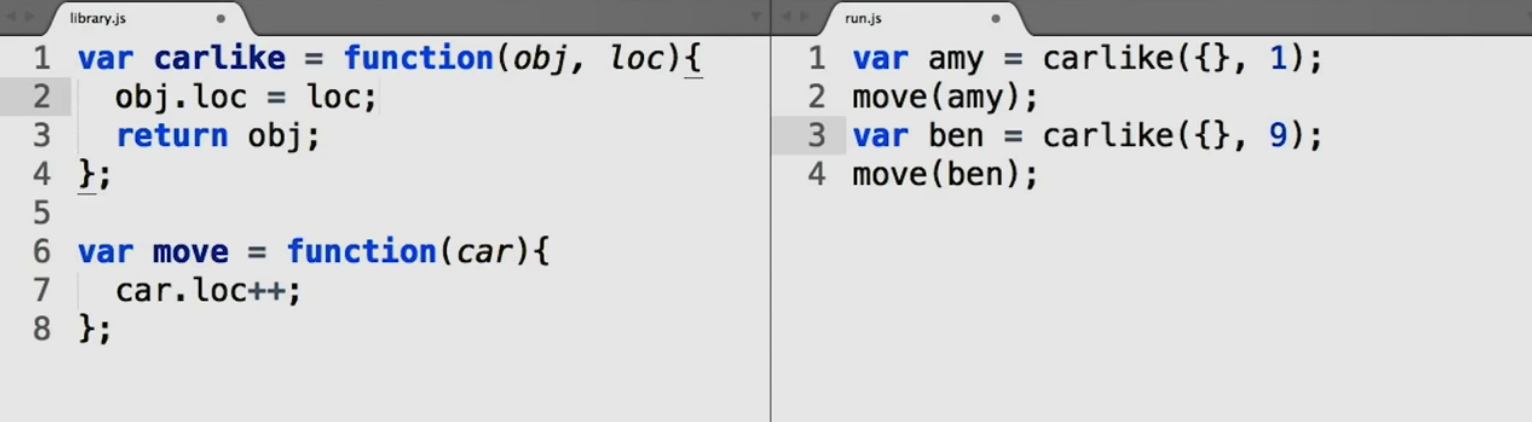
3.

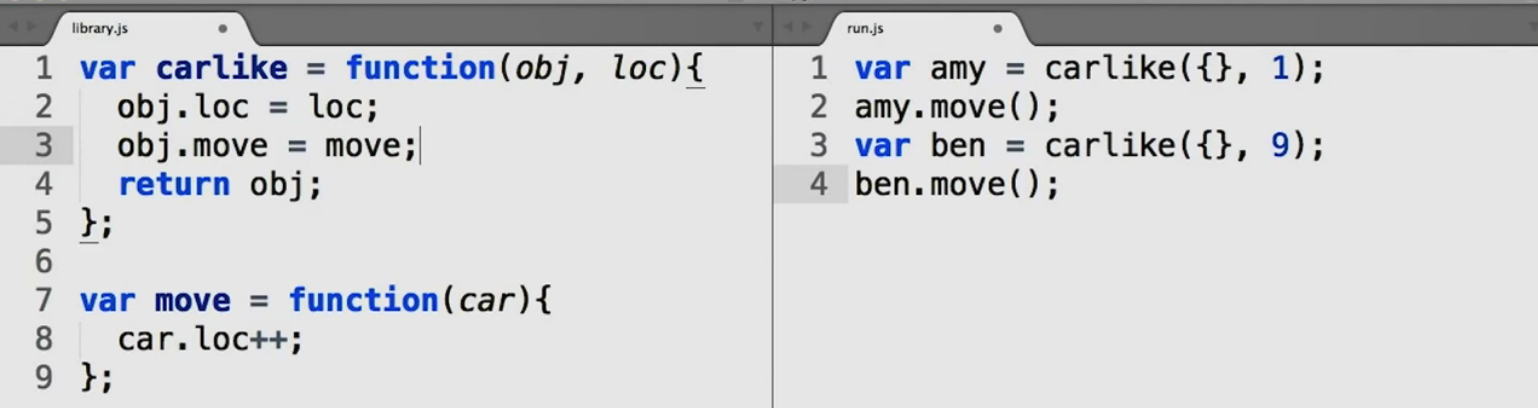
Prototype chains: this is useful technique for code sharing and for saving memory.

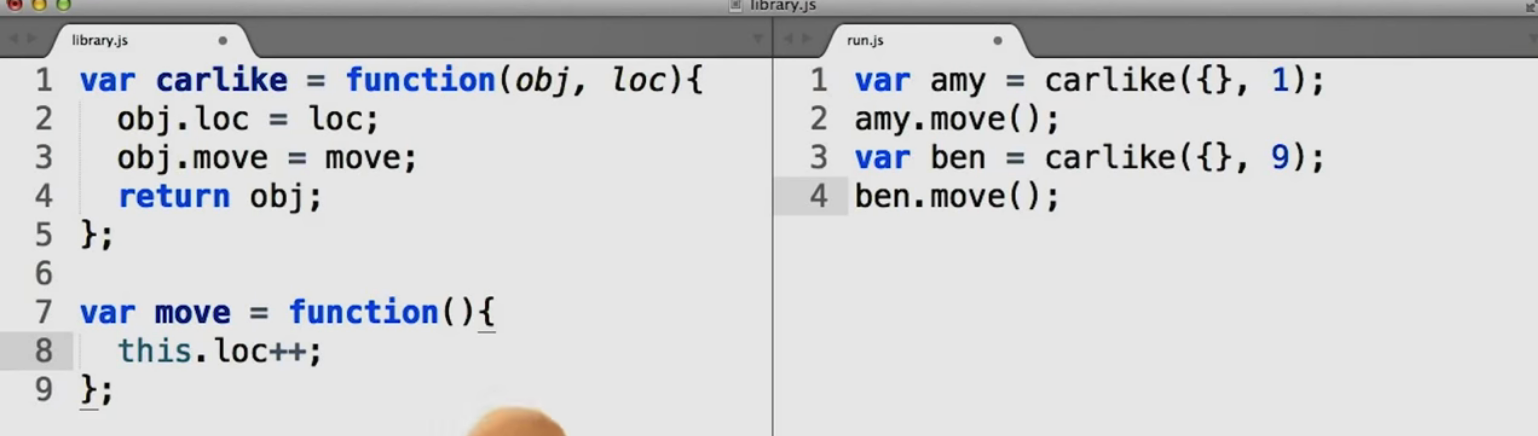


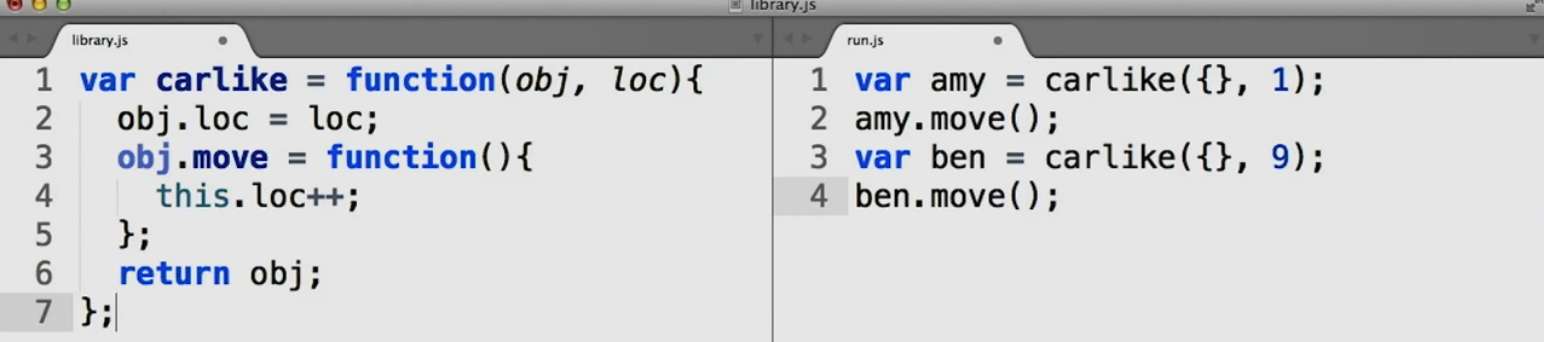
4. Object Decorator Pattern

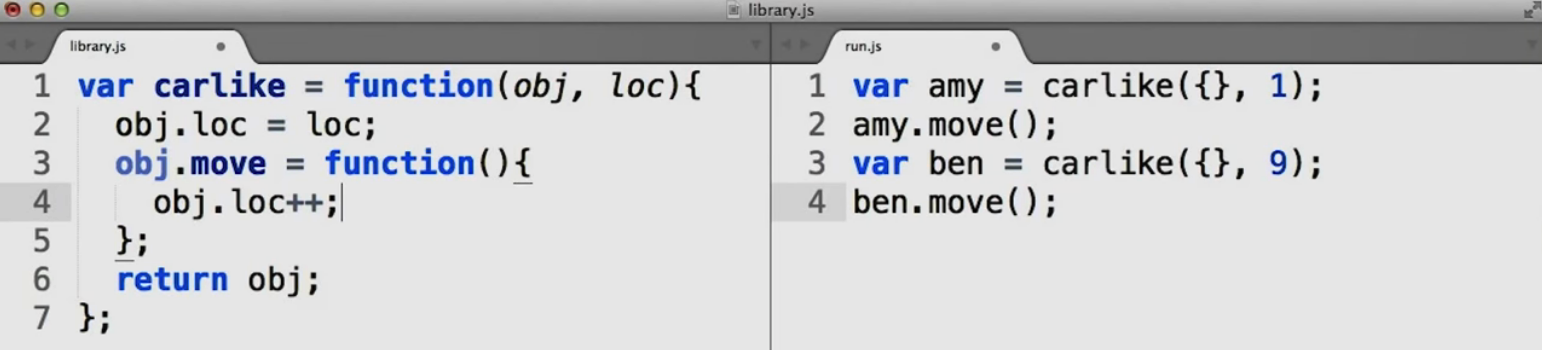
code reuse





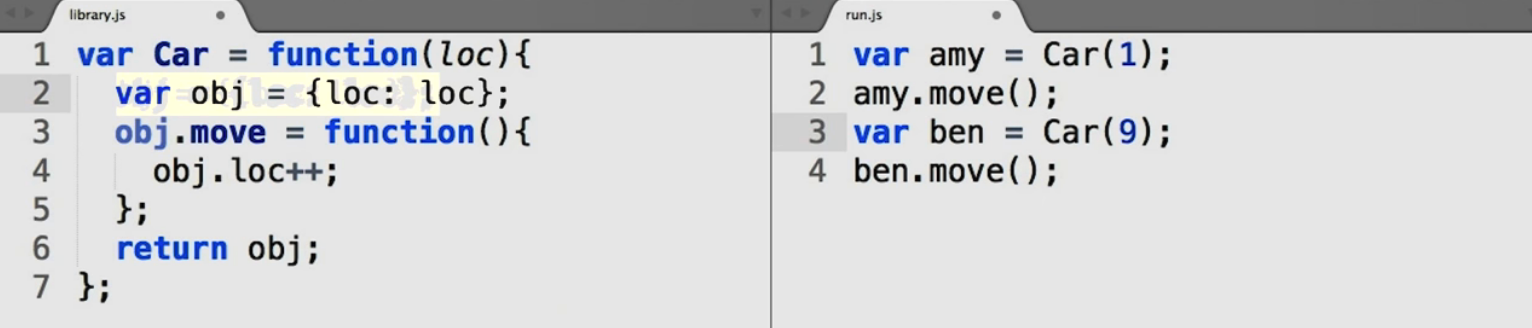






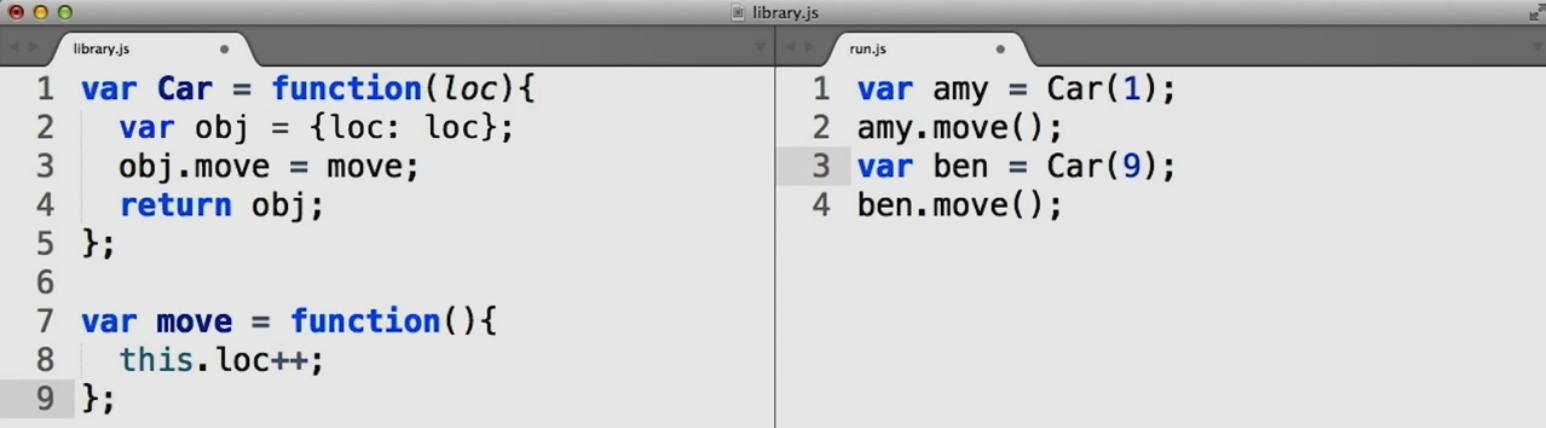
5.

function class pattern

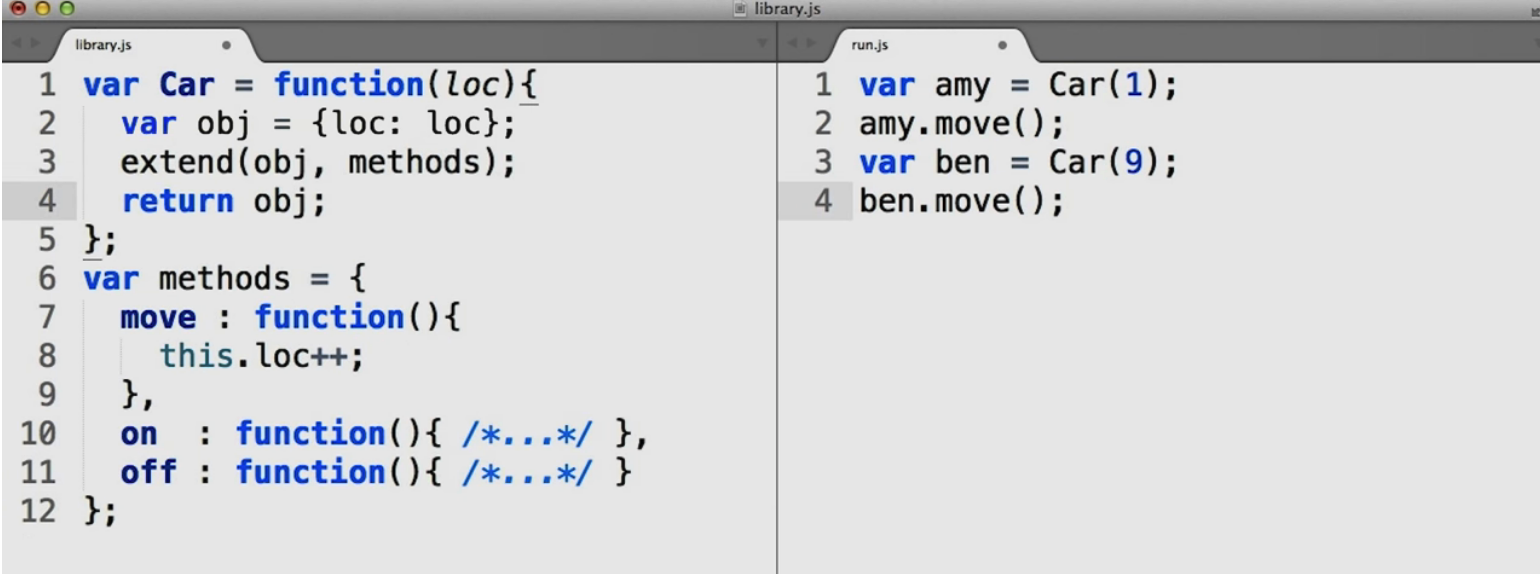


reduce duplicity:

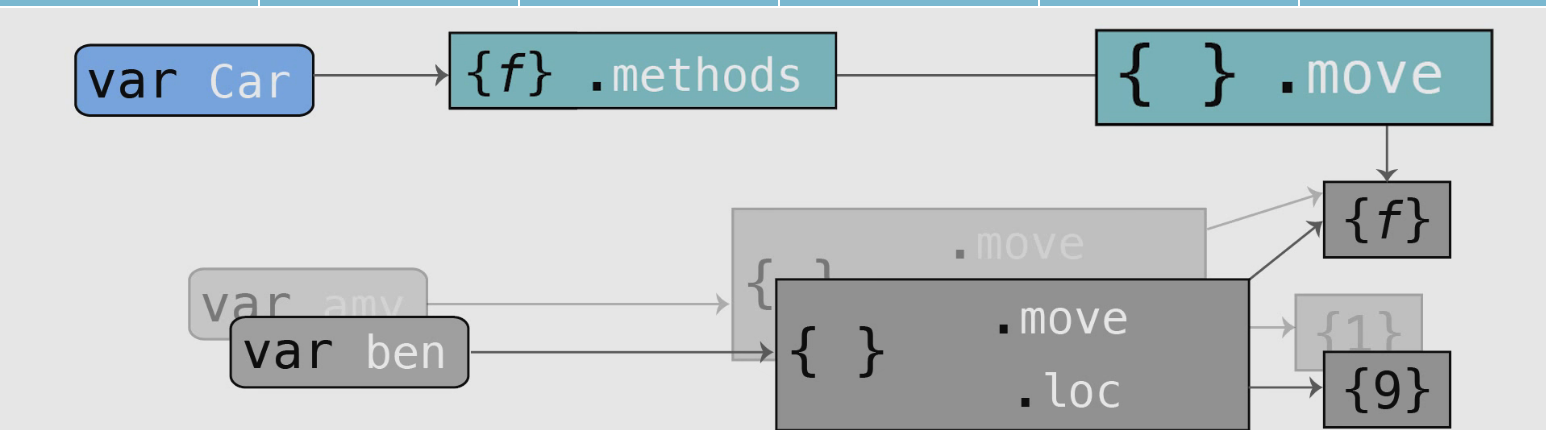
this replaces obj because this function is being called as a method on objects created withCar.



Functional shared patterns



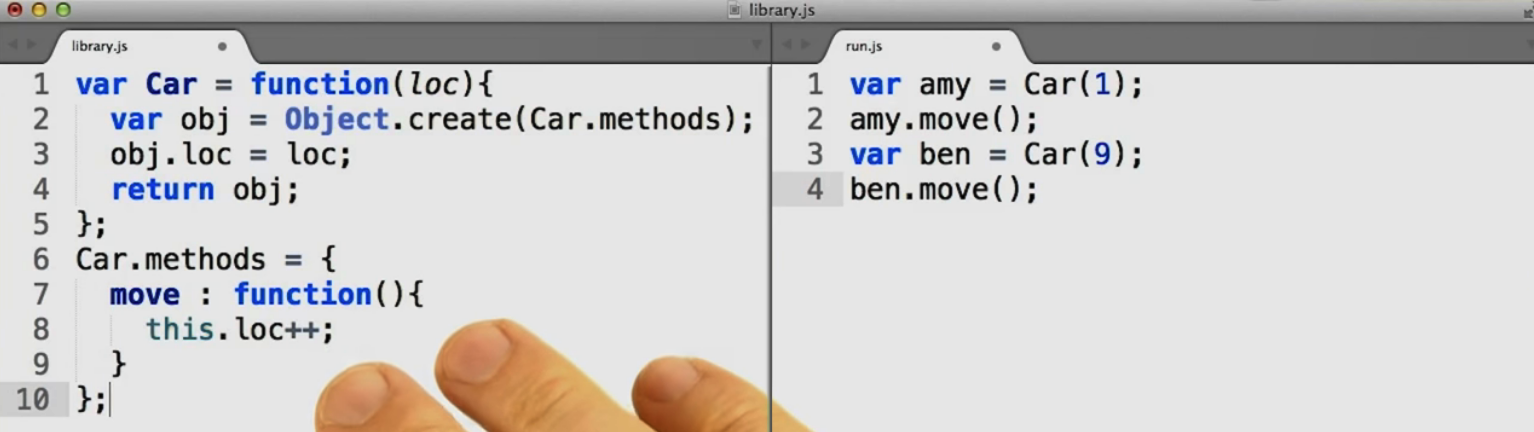




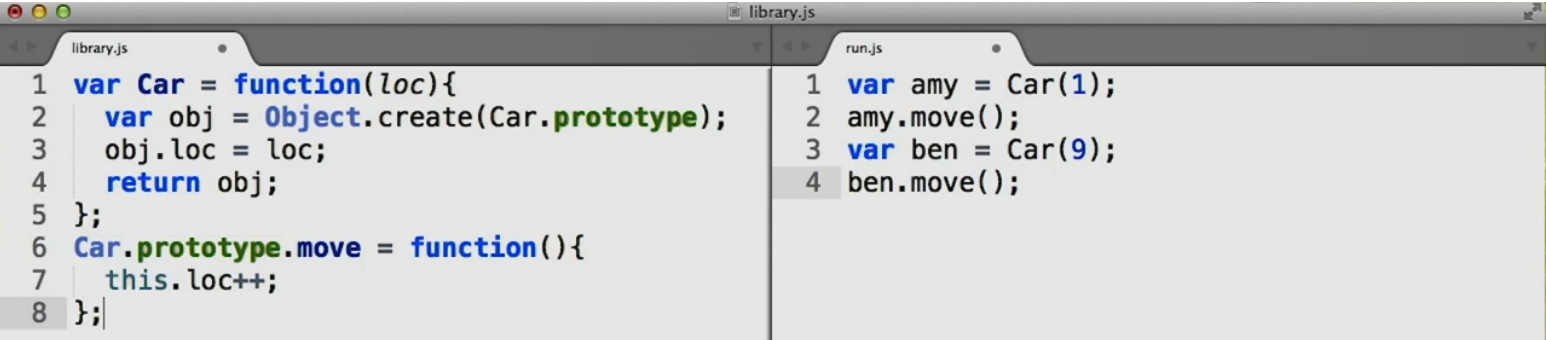
6.

prototype classes

prototypal pattern



Constructor Prototypes

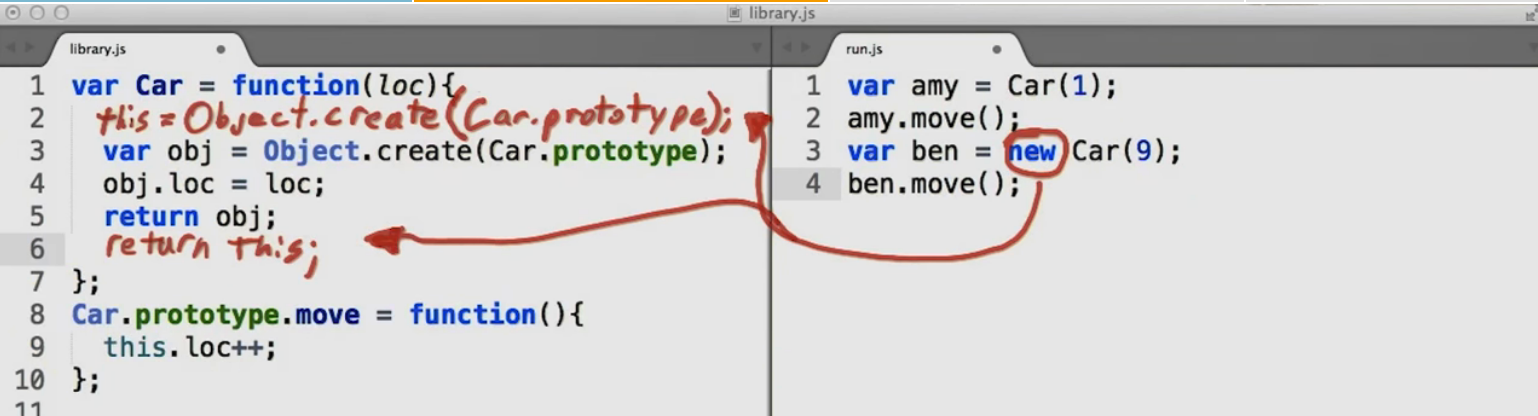


7.

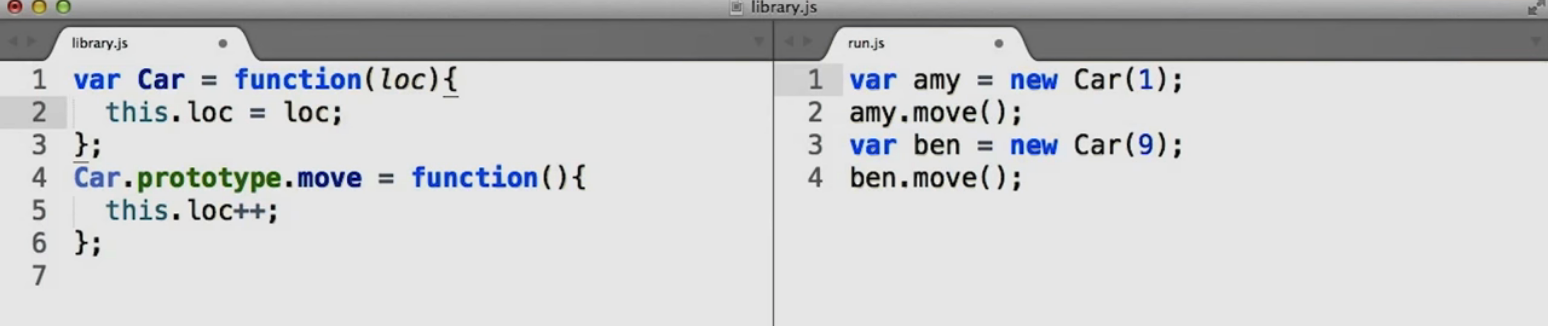
pseudocalssical patterns

Constructor Mode:Macintosh HD:Users:nali:Desktop:Snip20160204_16.png

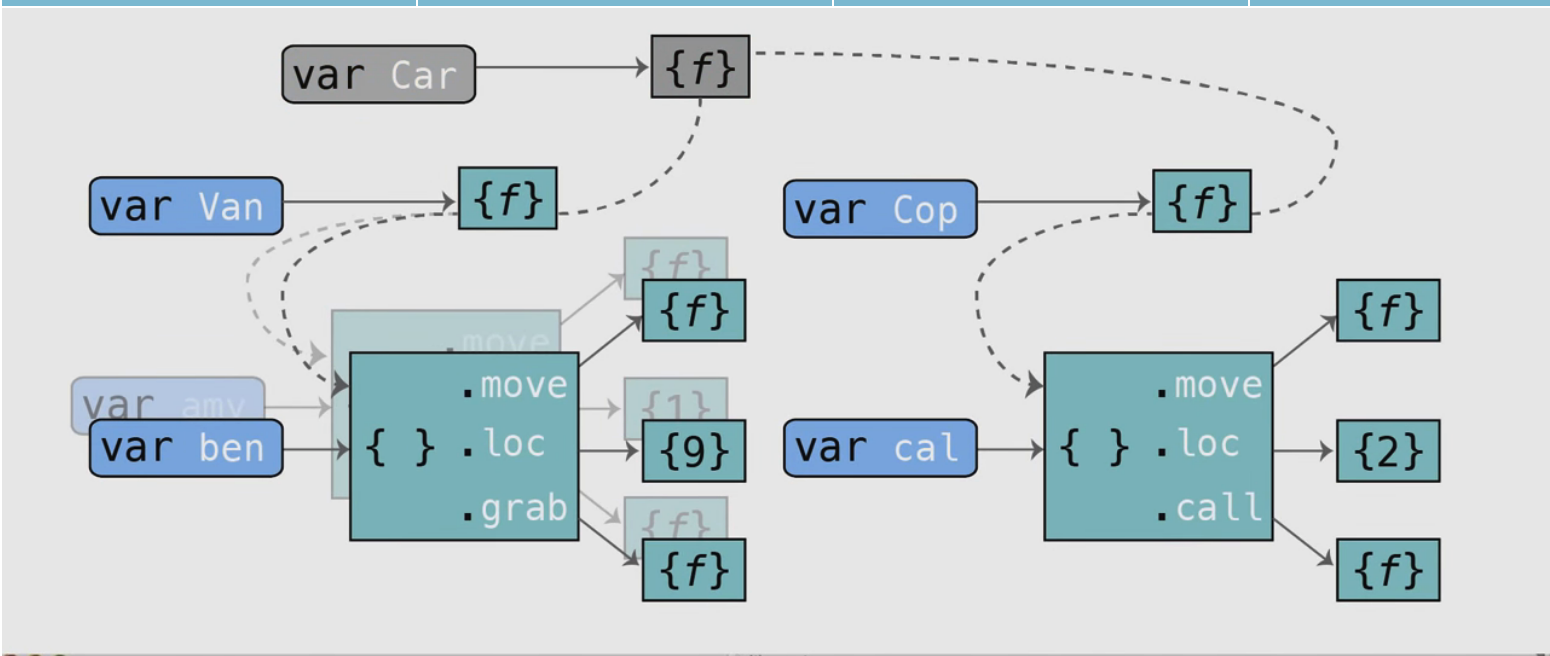
Macintosh HD:Users:nali:Desktop:Snip20160204_19.png



new provides us with the equivalent of lines 3 and 5 in our function, so we don't need them any more ... provided that we consistently use new!



8. Superclass and Subclass



use functional pattern

